5 CONCLUSIONS AND FUTURE WORK

In this thesis, we have presented a comprehensive suite of methods and tools for Software Diversification in WebAssembly. This chapter summarizes the technical contributions of this dissertation and the empirical findings of our research. Besides, we outline directions for future research.

5.1 Summary of technical contributions

We present three tools for Software Diversification in WebAssembly: CROW, MEWE and WASM-MUTATE. CROW is a compiler-based approach. It leverages enumerative synthesis to generate functionally equivalent code replacements and assembles them into diverse Wasm program variants. CROW uses SMT solvers to guarantee functional equivalence. MEWE provides dynamic execution path randomization by packaging variants generated out of CROW. Finally WASM-MUTATE, uses hand-made rewriting rules and random traversals over e-graphs to provide a binary-based solution for WebAssembly diversification.

TODO Wrapping up of the technical contributions and main challenges per tool. We provide all our tools

5.2 Summary of empirical findings

We have conducted a series of experiments to evaluate the effectiveness of our tools. In concrete we use WASM-MUTATE to demonstrate two scenarios of Software Diversification for WebAssembly: Offensive Software Diversification and Defensive Software Diversification.

Our work provides evidence that the malware detection community has opportunities to strengthen the automatic detection of cryptojacking

[0] Compilation probe time 2023/10/24 10:48:07

WebAssembly malware. The results of our work are actionable, as we also provide quantitative evidence on specific malware transformations on which detection methods can focus. The case discussed in this section is fully detailed in Cabrera-Arteaga et al. "WebAssembly Diversification for Malware Evasion"

WASM-MUTATE crafts WebAssembly binaries that are resilient to Spectre-like attacks. By integrating a software diversification layer into WebAssembly binaries deployed on Function-as-a-Service (FaaS) platforms, security can be significantly bolstered. This approach allows for the deployment of unique and diversified WebAssembly binaries, potentially utilizing a distinct variant for each cloud node, thereby enhancing the overall security. The case discussed in this section is fully detailed in Cabrera-Arteaga et al. "WASM-MUTATE: Fast and Effective Binary Diversification for WebAssembly"

TODO Overall results for Offensive **TODO** Overall results for Defensive

TODO General thoughts on the results

Although our experiments are conducted prior to submitting the WebAssembly binary to the FaaS platform, we argue that WebAssembly binary diversification could be implemented at any stage of the FaaS workflow. The same argument holds by using any other diversification tool presented in this dissertation (see Chapter 3).

5.3 Future Work

Along with this dissertation we have highlighted several open challenges related to Software Diversification in WebAssembly. These challenges open up several directions for future research. In the following, we outline some of these directions.

Extending WASM-MUTATE: WASM-MUTATE may gain advantages from the enumerative synthesis techniques employed by CROW and MEWE. Specifically, WASM-MUTATE could adopt the transformations generated by these tools as rewriting rules. This approach could enhance WASM-MUTATE in two specific ways. Firstly, it could improve the preservation of the variants generated by WASM-MUTATE. The primary reason for this is that enumerative synthesis outperforms compiler optimizations, making it more challenging for reverse engineers to identify the original program. Secondly, this method would inevitably expand the diversification space of WASM-MUTATE e-graphs.

Program Normalization: We successfully employed WASM-MUTATE for the evasion of malware detection (see Section 4.1). The proposed mitigation in the prior study involved code normalization as a means of reducing the spectrum of malware variants. Our current work provides insights into the potential effectiveness of this approach. Specifically, a practically costless process of pre-compiling Wasm binaries could be employed as a preparatory measure for malware classifiers. In other words, a Wasm binary can first be JITed to machine

code, effectively eliminating approx. 24% of malware variants according to our preservation statistics highlighted in Table 3.1. This approach could substantially enhance the efficiency and precision of malware detection systems.

Fuzzing WebAssembly compilers Generating well-formed inputs in fuzzing campaigns presents a significant challenge [**?**]. This challenge is particularly prevalent in fuzzing compilers, where the inputs need to be executable yet complex enough programs to test various compiler components. To address this issue, our tools can generate semantically equivalent variants from an original Wasm binary, thereby improving the scope and efficiency of the fuzzing process. A practical instance of the effectiveness of this approach occurred in 2021, when it led to the discovery of a security CVE in wasmtime [**?**].

Mitigating Port Contention Rokicki et al. [**?**] demonstrated the feasibility of a covert side-channel attack using port contention within WebAssembly code in the browser. This attack essentially depends on the precise prediction of Wasm instructions that trigger port contention. To address this security issue, one could conveniently implement Software Diversification as a browser plugin. Software Diversification has the capability to substitute the Wasm instructions used as port contention predictors with other instructions. This is very similar to the effect on timers and padding previously discussed in Section 4.3. Such an approach would inevitably eliminate the port contention in the specific port used for the attack, thereby strengthening browsers against such harmful actions.

Dataset augmentation: The WebAssembly ecosystem is still in its infancy compared to more mature programming environments. The study by Hilbig et al. in 2021 found only 8,000 unique WebAssembly binaries globally[**?**], a fraction of the 1.5 million and 1.7 million packages available in npm and PyPI, respectively. This limited dataset poses challenges for machine learning-based analysis tools, which require extensive data for effective training. The scarcity of WebAssembly programs also exacerbates the problem of software monoculture, increasing the risk of compromised WebAssembly programs being consumed[**?**].

AI and Software Diversification As we discussed in Chapter 3, having a diversifier at the high language level seems impractical due to the wide range of existing frontends. Wit the appearing of Large Language Models and their accuracy in generating high-level language such issue could be solved. Yet, we believe it is not only a matter of connecting the LLM to the diversifier, studies on preservation should be acompilshed. In concrete, high-level diversification migth lead to low preservation, making false the assumption of diversification at low-level.